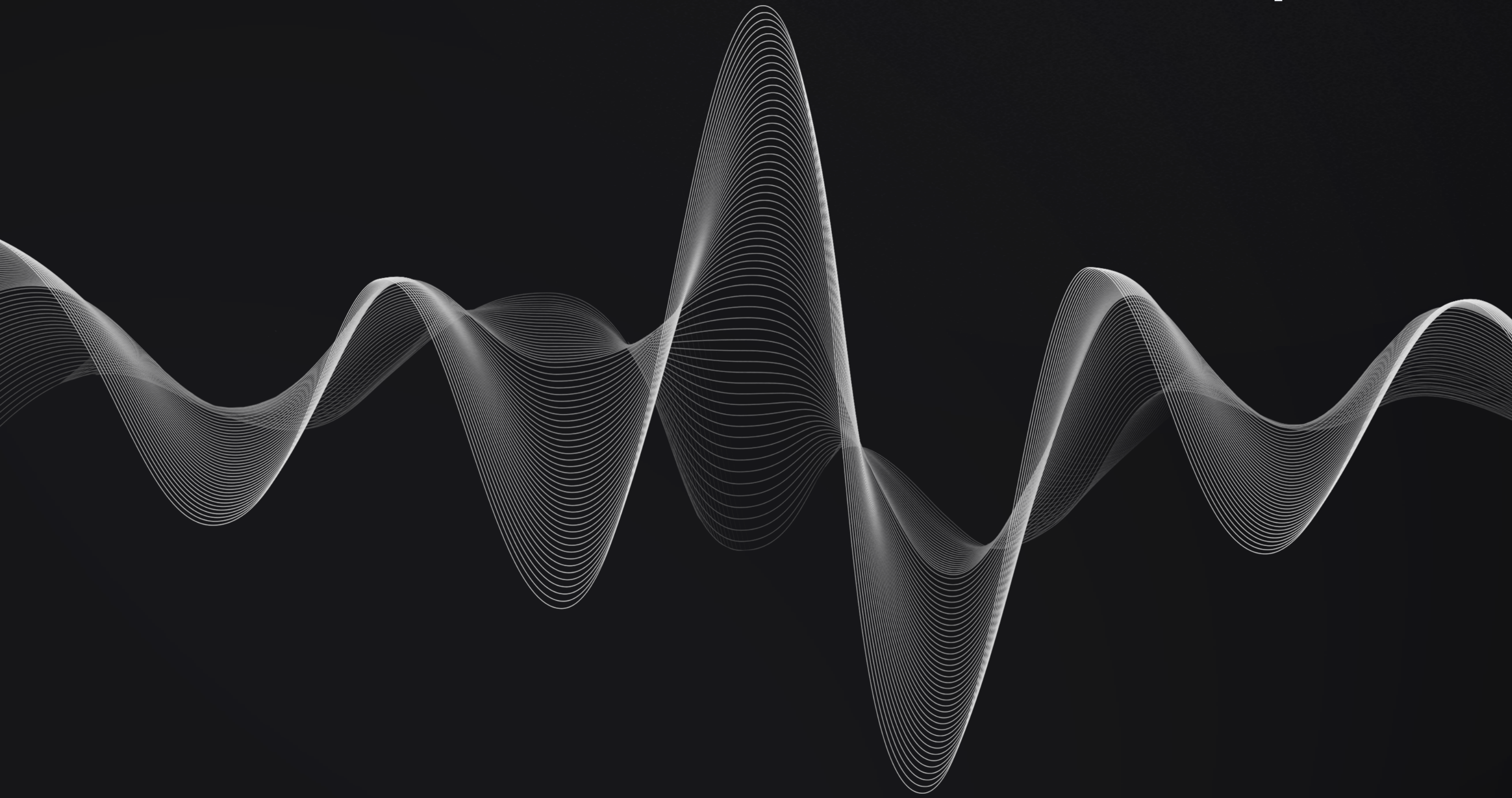
RESONANCE

nodle

Nodle

Nodle Staking Smart Contract Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

PUBLIC **FINAL**(v2.1)

Audit_Report_NODL-STK_FINAL_21

Jun 10, 2025	○	v0.1	Luis Arroyo: Initial draft
Jun 11, 2025	○	v0.2	Luis Arroyo: Added findings
Jun 13, 2025	🟢	v1.0	Charles Dray: Approved
Jun 24, 2025	○	v1.1	João Simões: Reviewed findings
Jun 25, 2025	🟡	v2.0	Charles Dray: Finalized
Jul 11, 2025	○	v2.1	Charles Dray: Published

Points of Contact	Micha Benoliel	Nodle	micha@nodle.com
	Charles Dray	Resonance	charles@resonance.security
Testing Team	Luis Arroyo	Resonance	luis.arroyo@resonance.security
	João Simões	Resonance	joao@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security
	Ilan Abitbol	Resonance	ilan@resonance.security

Copyright and Disclaimer

Contents

Executive Summary

Nodle contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between June 11, 2025 and June 13, 2025. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 3 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Nodle with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

System Overview

The Nodle Staking contract implements a NODL staking mechanism that yields rewards to the users after a certain duration of time. Users are able to stake multiple times, unstake, and claim token rewards. The contract is access controlled and implements emergency pausability through its administrative user roles. The staking mechanism is implemented for the ZKSync blockchain.

Repository Coverage and Quality

Code	**Tests**	**Documentation**
8 / 10	9 / 10	7 / 10

Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is good**.
- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is 98%. Overall, **tests coverage and quality is excellent**.
- The documentation includes technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is good**.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: `NodleCode/rollup`
- Hash: 44bb5290f48f37915c765fbc96c17bb79bad1dd1

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak **-** Inconsequential or minimal damage or loss
2. Medium **-** Temporary or partial damage or loss
3. Strong **-** Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely **-** Requires substantial knowledge or effort or uncontrollable conditions
2. Likely **-** Requires technical knowledge or no special conditions
3. Very Likely **-** Requires trivial knowledge or effort or no conditions

			Likelihood	
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- ıııııı **"Quick Win"** Requires little work for a high impact on risk reduction.
- ıılıı **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ılııl **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	🟧	Insufficient Rewards Can Prevent Users From Claiming Or Unstaking	ıılıı	Resolved
RES-02	🟧	emergencyWithdraw() May Cause Denial of Service	ıılıı	Resolved
RES-03	🟩	index In claim() And unstake() Should Be Controlled	ıılıı	Resolved
RES-04	🟩	Precision Loss In Rewards	ıılıı	Resolved
RES-05	🟩	Centralization Risk	ılııl	Acknowledged
RES-06	🟩	Misleading Setting Of claimed Variable	ıııııı	Resolved
RES-07	🟩	Rewards Can Be Fund By Users	ıılıı	Resolved
RES-08	⬜	uint Should Not Be Initialized With 0	ıılıı	Resolved
RES-09	⬜	storage Variable Should Be Changed To memory	ıılıı	Resolved

Insufficient Rewards Can Prevent Users From Claiming Or Unstaking

Medium **RES-NODL-STK01** Business Logic **Resolved**

Code Section

- `Staking.sol#L155`

Description

The `InsufficientRewardBalance` check only verifies if there are enough tokens at the moment the function is called. It does not account for the possibility that other users might claim rewards or unstake concurrently, depleting the balance before transactions are processed.

this can be replicated in this scenario:

- `user1` and `user2` put 10 tokens each one.
- Rewards are 20%, so each one will receive 12 tokens.
- Staking contract has 1 reward token. Pool now contains a total of 21 tokens.
- `user1` claims and obtain 12 tokens. Pool now contains a total of 9 tokens.
- `user2` claims or unstakes but their transaction gets reverted because there is not enough tokens in the smart contract.

Recommendation

It is recommended to only allow users to stake new tokens if there are enough funds within the smart contract to cover both the stake value and the potential rewards value. Otherwise, the balance of NODL tokens inside the smart contract should be increased before stake durations finish, in order to accommodate every possible user's funds and their rewards.

Status

The issue has been fixed in e523bc648b1b10f621fc38a95a64fbe0199dc2b9.

emergencyWithdraw() May Cause Denial of Service

Medium **RES-NODL-STK02** Data Validation **Resolved**

Code Section

- `Staking.sol#L172`

Description

The `emergencyWithdraw()` function forcefully withdraws all tokens from the contract and sets `totalStakedInPool = 0`. Users attempting to claim rewards or unstake after `emergencyWithdraw()` will encounter issues since `totalStakedInPool` is already 0 and reverts the transaction. In addition, if owner adds the funds again, the value will remain 0, blocking the smart contract of future transactions.

Recommendation

It is recommended to remove the `totalStakedInPool = 0` from the `emergencyWithdraw()` function.

Status

The issue has been fixed in e523bc648b1b10f621fc38a95a64fbe0199dc2b9.

index In claim() And unstake() Should Be Controlled

Low **RES-NODL-STK03** Data Validation **Resolved**

Code Section

- `Staking.sol#L148`
- `Staking.sol#L193`

Description

The `claim()` and the `unstake()` functions take an `index` as input, but there's no check to ensure the index is within the bounds of the `stakes[msg.sender]` array before accessing the array.

Recommendation

It is recommended to add a check to ensure `index < stakes[msg.sender].length` at the beginning of the function.

Status

The issue has been fixed in e523bc648b1b10f621fc38a95a64fbe0199dc2b9.

Precision Loss In Rewards

Low

RES-NODL-STK04 Data Validation **Resolved**

Code Section

- `Staking.sol#L153`

Description

The reward calculation suffers from potential precision loss due to integer division. Small stakes or low reward rates could result in users receiving less reward than expected.

Recommendation

It is recommended to use a higher precision calculation like scaling up values before division.

Status

The issue has been fixed in e523bc648b1b10f621fc38a95a64fbe0199dc2b9.

Centralization Risk

Low **RES-NODL-STK05** Governance **Acknowledged**

Code Section

- Not specified

Description

The smart contract contains several functions access controlled by administrative users with privileged rights in charge of performing admin tasks such as pausing and withdrawing funds during emergencies. These users need to be trusted not to perform malicious updates on the contract.

Recommendation

It is recommended to implement solutions like a multi-signature wallet to distribute admin control among multiple trusted parties. This ensures that critical actions can only be executed if a predefined quorum of trusted parties approves the action, reducing the risk of unilateral decisions or key compromise.

Another possible solution is to implement a decentralized party that handles administrative functions, for example with the implementation of DAO solutions.

Status

The issue was acknowledged by Nodle's team. The development team stated "It's planned to use the same wallet as for other previous contracts which is already a multisig, we wanted to keep it simple".

Misleading Setting Of claimed Variable

Low

RES-NODL-STK06 | Business Logic | **Resolved**

Code Section

- `Staking.sol#L202`

Description

The function `unstake()` allows users to desist from their stake, getting their funds back to their accounts. In technical terms, with this action, funds are never claimed but are instead unstaked. As such, the `claimed` variable of the user's stake should be left as `false`, to correctly show users of the platform of the action that was taken, e.g. when calling the function `getStakeInfo()` with the index of the specific stake.

Recommendation

It is recommended to refrain from setting the claimed variable to `true` when executing the function `unstake()`, as it may be misleading.

Status

The issue has been fixed in e523bc648b1b10f621fc38a95a64fbe0199dc2b9.

Rewards Can Be Fund By Users

Low **RES-NODL-STK07** Access Control **Resolved**

Code Section

- `Staking.sol#L136`

Description

The `fundRewards()` function is intended to be used only by `REWARDS_MANAGER_ROLE`but it is possible for any user to transfer `NODL` tokens to the contract and increase the rewards.

Recommendation

It is recommended to use an internal variable to keep track of the amount of available rewards.

Status

The issue has been fixed in e523bc648b1b10f621fc38a95a64fbe0199dc2b9.

uint Should Not Be Initialized With 0

Info **RES-NODL-STK08** Gas Optimization **Resolved**

Code Section

- `Staking.sol#L237`
- `Staking.sol#L238`

Description

Initializing variables to their default values spends gas unnecessarily.

Recommendation

It is recommended to not initialize variables to their default values, in this case 0.

Status

The issue has been fixed in e523bc648b1b10f621fc38a95a64fbe0199dc2b9.

storage Variable Should Be Changed To memory

Info **RES-NODL-STK09** Gas Optimization **Resolved**

Code Section

- `Staking.sol#L233`

Description

The `getStakeInfo()` function is declared as `view`, which means it should not modify the contract's state. Using a `storage` pointer would imply a potential modification of the state, which is not allowed in a `view` function. Storage variables also consume more gas.

Recommendation

Changing `s` to `memory` can potentially save gas because it avoids reading from storage. It is safe to not use variables that can change state in a `view` function.

Status

The issue has been fixed in e523bc648b1b10f621fc38a95a64fbe0199dc2b9.

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.